PetaQCD : En Route for the automatic code generation for lattice QCD

D. Barthou, G. Grosdidier, Christine Eisenbeis, P. Guichon, M. Kruse, Olivier Pene, Konstantin Petrov, Claude Tadonki

▶ **To cite this version:**

D. Barthou, G. Grosdidier, Christine Eisenbeis, P. Guichon, M. Kruse, et al.. PetaQCD : En Route for the automatic code generation for lattice QCD. Lattice 2011, Jul 2011, Squaw Valley, Lake Tahoe, United States. pp.043. hal-00833330

HAL Id: hal-00833330
https://hal.science/hal-00833330

Submitted on 12 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PROCEEDINGS OF SCIENCE

PetaQCD: En Route for the Automatic Code Generation for Lattice QCD

Denis Barthou
University of Bordeaux
351 cours de la Liberation, 33405 Talence Cedex

Gilbert Grosdidier

LAL/IN2P3/CNRS 91405 Orsay Cedex

Christine Eisenbeis

INRIA, 91405 Orsay Cedex

Pierre Guichon

IRFU/CEA-Saclay, 91191 Gif-sur-Ivette

Michael Kruse

INRIA, 91405 Orsay Cedex

Olivier Pene

LPT/CNRS 91405 Orsay Cedex

Konstantin Petrov*
IRFU/CEA-Saclay, 91191 Gif-sur-Ivette
E-mail: konstantin.petrov@cea.fr

Claude Tadonki

LAL/CNRS et Ecole des Mines, 60, Boulevard Saint-Michel 75272, Paris Cedex 06

New computer architectures with various weak and strong characteristics appear with increasing speed. We present our work in progress for the tool-chain aimed at rapid prototyping of the novel dirac matrix inversion algorithms for emerging architectures. From scientific description of the algorithm on the front end to the several back ends we discuss how symbolic manipulation may be used to create and optimize lattice calculations on the fly.

The XXIX International Symposium on Lattice Field Theory - Lattice 2011
July 10-16, 2011
Squaw Valley, Lake Tahoe, California

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(Lattice 2011)043

1. Introduction

Since the early days of Lattice QCD calculations we often find ourselves in a situation when only a very short period of time is granted before the announcement of a novel architecture and its deployment in the computing centers across the globe. Commercial compilers often take years to catch up and the resulting punishment in terms of effectiveness is often a factor of three to ten. It may even happen that a given arhitecture is cancelled before the very-high-performance library appears, see e.g. [1]. Since a couple of years a collaboration which aims to improve this situation this situation has emerged [2]. We plan to do so by producing a tool chain which will give us a short-cut between high-level mathematical description and the high-performance executable code. Here we report on the progress in our work.

2. High level description

To start the code generation we need some sort of high-level mathematical language to describe our Lagrangians and Algorithms. As it is natural for scientists to use Latex for typesetting formulae, we utilize it as well for the high-level description. With suitable headers such description may be compiled using any modern Latex system to produce complete visualisation of the scientific foundations for the inversion problem. However Latex as such does not participate in the code generation, instead a symbolic manipulation (currently implemented in Maude language) is used. Let us consider an example for a typical Lattice QCD problem. We start with the definitions for our operator of choice, Twisted Mass Wilson Dirac:

```
\begin{definition}{Dirac}
\caption{Definition of Dirac matrix on a Lattice $L$,
of gauge matrices and of the preconditioners}
\Constant{$\Dirac, \Pe, \Po, \gamma_5 , \iQ \in M,
\Preconditionera, \Preconditionerb \in M -> M,
\gamma, \sigma ,\ga \in Index -> M, U \in Index -> Index -> M,
\kappa, \mu, \epsilon \in \Complex,
D \in Indexset, dx, dy, dz, dt \in Index$}
```

Then we declare what is input and what is output:

```
\Input{$c \in \Complex, d1\in Index, s1\in Index$}
\Var{$s \in Index,  d \in Index$}
```

and the dirac operator can then be expressed as:

```
 \begin{align}
 \Dirac &=
 \id{L \tensor C \tensor S} \\
```


PoS(Lattice 2011)043

```
&+ ( (2 * i * \kappa * \mu) * (\id{L} \tensor \id{C} \tensor \gamma_5)
 )& +  \ssum{d \in  D}{(\shift{L}{d} \tensor \id{C \tensor S}) *
 \dsum{s\in L}
{ \U(d,s) \tensor (\id{S} + \gamma(d))}}
 & + \ssum{d \in  D}{(\shift{L}{- d} \tensor \id{C \tensor S}) *
 \dsum{s\in L}
{ \U(- d,s) \tensor (\id{S} - \gamma(d))}} \;\\
```

Finally, we specify the even-odd preconditioner:

```
 \Pe &= \pr{even}{L} \tensor \id{C \tensor S} \;\\
 \Po &= \pr{!even}{L} \tensor \id{C \tensor S}\;
 \end{align}
```

Once compiled, it of course looks much more readable:

Constant: $Dirac, P_e, P_o, \gamma_5, iQ \in M, \text{Preconditioner1}(,)\text{Preconditioner2}(\in)M->$
$M, \gamma, \sigma, g_a \in Index-> M, U \in Index-> Index-> M, \kappa, \mu, \epsilon \in \mathbb{C}, D \in$
$Indexset, dx, dy, dz, dt \in Index$
Input : $c \in \mathbb{C}, d1 \in Index, s1 \in Index$
Var : $s \in Index, d \in Index$

$$
\begin{aligned}
Dirac = & I_{L\otimes C\otimes S} \\
& + ((2 * i * \kappa * \mu) * (I_L \otimes I_C \otimes \gamma_5)) \\
& + \sum_{d\in D} (J_L^d \otimes I_{C\otimes S}) * \bigoplus_{s\in L} U(d,s) \otimes (I_S + \gamma(d)) \\
& + \sum_{d\in D} (J_L^{-d} \otimes I_{C\otimes S}) * \bigoplus_{s\in L} U(-d,s) \otimes (I_S - \gamma(d)) \\
P_e = & P_{even,L} \otimes I_{C\otimes S} \\
P_o = & P_{!even,L} \otimes I_{C\otimes S}
\end{aligned}
$$

The variables we listed are not independent and some are linked while some, like gamma matrices, are almost fixed. So we will represent such constraints via following relation and identities:

```
\begin{align}
\U(- d1,s1) &= \U(d1,s1 - d1)^\dagger \; \\
\Preconditionera{Dirac} & = Pe \;  \\
\Preconditionerb{Dirac} & = Po \;  \\
\gamma( d1)^\dagger & = - \gamma(d1) \; \\
\sigma(- d1) & = - \sigma(d1) \; \\
\ga(d1) & = i * \gamma_5 * \gamma(d1) \; \\
\isInvertible{\id{S} + c * \gamma_5} & = true \; \\
```


PoS(Lattice 2011)043

```
\isInvertible{\id{S} - c * \gamma_5} & = true \; \\
\type{\gamma(d1)} & = \matrixtype{S}{S} \; \\
\type{\sigma(d1)} & = \matrixtype{HalfS}{HalfS} \; \\
\type{\U(d1,s1)} & = \matrixtype{C}{C} \; \\
\type{\gamma_5} & = \matrixtype{S}{S} \;  \\
\gamma_5^\dagger & = \gamma_5 \; \\
D &= \{ dx, dy, dz, dt\} \; \\
```

Or, in human language, it will look like this:

$$
\begin{aligned}
U(-d1, s1) &= U(d1, s1 - d1)^{\dagger} \\
\text{Preconditioner1}(Dirac) &= Pe \\
\text{Preconditioner2}(Dirac) &= Po \\
\gamma(d1)^{\dagger} &= -\gamma(d1) \\
\sigma(-d1) &= -\sigma(d1) \\
g_a(d1) &= i * \gamma_5 * \gamma(d1) \\
\text{invertible}(I_S + c * \gamma_5) &= true \\
\text{invertible}(I_S - c * \gamma_5) &= true \\
\text{type}(\gamma(d1)) &= S \times S \\
\text{type}(\sigma(d1)) &= HalfS \times HalfS \\
\text{type}(U(d1, s1)) &= C \times C \\
\text{type}(\gamma_5) &= S \times S \\
\gamma_5^{\dagger} &= \gamma_5 \\
D &= \{dx, dy, dz, dt\}
\end{aligned}
$$

The following template is one example of the iterative methods that can be applied to solve our problem. Conjugate gradient is one of the most used methods, due to its effectiveness and universality.

```
 \begin{template}{CG}
\Input{$A\in M , b \in V$}
\Output{$x \in V$}
\Match{$x = A^{-1} * b \;$}
\Require{$A^\dagger = A $}
\Var{$r, r_1, Ap,p \in V , \alpha, \beta, n_r, n_{r1}\in \Complex$}
$r = b$ \;
$p = r$ \;
$n_r = (r \hdp r)$ \;
\While{$(n_r > \epsilon$)}{
```


PoS(Lattice 2011)043

```
  $Ap = A * p$ \;
  $n_r = (r \hdp r)$ \;
  $\alpha = n_r / (p \hdp Ap)$ \;
  $x  = x + \alpha * p$ \;
  $r = r - \alpha * Ap$ \;
  $n_{r1} = (r \hdp r)$ \;
  $\beta = n_{r1} / n_r$ \;
  $p = r + \beta * p$ \;
  $n_r = n_{r1}$ \;
}
\caption{Conjugate Gradient [CG]}
\end{template}
```

Such code will be directly compiled by Latex into:

$$
\begin{aligned}
U(-d1, s1) &= U(d1, s1 - d1)^{\dagger} \\
\text{Preconditioner1}(Dirac) &= Pe \\
\text{Preconditioner2}(Dirac) &= Po \\
\gamma(d1)^{\dagger} &= -\gamma(d1) \\
\sigma(-d1) &= -\sigma(d1) \\
g_a(d1) &= i * \gamma_5 * \gamma(d1) \\
\text{invertible}(I_S + c * \gamma_5) &= true \\
\text{invertible}(I_S - c * \gamma_5) &= true \\
\text{type}(\gamma(d1)) &= S \times S \\
\text{type}(\sigma(d1)) &= HalfS \times HalfS \\
\text{type}(U(d1, s1)) &= C \times C \\
\text{type}(\gamma_5) &= S \times S \\
\gamma_5^{\dagger} &= \gamma_5 \\
D &= \{dx, dy, dz, dt\}
\end{aligned}
$$

We can typset any sufficiently simple algorithm, e.g. GCR, NRGCR etc in this manner and it will likely result in the converging code. For more complicated algorithms, like deflation, or any general Schur compliment method, the work is in progress.

PoS(Lattice 2011)043

3. Pseudocode generation

Once we decided on the algorithm and lagrangian, we can do the first step, the generation of the pesudocode. This can be done in two distinct manners, the microscopic and the macroscopic. The former is intended, primarily, for the validation of the algorithm although it may also be used for the initial study. Here is how is usually looks like, a snippet from the Conjugate Gradient implementation.

```
 for(j in {j | j < nrestart and nr > epsilon})
  {
    alpha = (Mp[j][L].Mp[j][L]) ^-1 * (r[L].Mp[j][L]);
     forall(x = 0 ; x < N ; x+=1)
      forall(y = 0 ; y < N ; y+=1)
       forall(z = 0 ; z < N ; z+=1)
        forall(t = 0 ; t < N ; t+=1)
                 {
       xx[x][y][z][t] = alpha*P[j][x][y][z][t]+xx[x][y][z][t];
       r[x][y][z][t] = r[x][y][z][t]-alpha*Mp[j][x][y][z][t];
       Mr[j+1] = (id(C) x (id(S)+ i*mu*kappa*2*gamma5)
+ sum(d in {dx,dy,dz,dt})(U(d, s-d-d)^* x (-gamma(d) + id(S)))
+ sum(d in {dx,dy,dz,dt})(U(d, d+s) x (id(S) + gamma(d))))
                      * r[x][y][z][t] ;}
for(i in 0 .. j)
{
delta[i] = -((Mp[i][L].Mp[i][L])^-1 * (Mr[j + 1][L].Mp[i][L]));
}
```

From here another code generation system, which we call Lion, is used to produce executable C code, adding proper headers and declarations. This has been completed and verified.

The second option, the macroscopic way, is rather intended for the high-performance implementation. The pseudocode looks similar, but without loops, for an easy mapping to the pre-optimised library functions. The resulting performance will depend on two things, the effectiveness of the hopping matrix multiplication and the speed of the underlying linear algebra library. This part of the projet is under heavy development.

References

[1] An efficient CELL library for Lattice Quantum Chromodynamics., Claude Tadonki, Gilbert Grosdidier, Olivier Pene, International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART) , Tsukuba, Japan, June 1-4, 2010

[2] Towards the Petaflop for Lattice QCD Simulations: the PetaQCD Project, Jean-Christian Anglès d'Auriac et al., 2010 J. Phys.: Conf. Ser. 219 052021

[3] Maude, http://maude.cs.uiuc.edu/

PoS(Lattice 2011)043